# 自底语法分析

2020年7月1日 21:13

1. 自底向上的语法分析：从分析树的底部(叶节点)向顶部(根节点)方向构造分析树
    a. 可以看成是将输入串w归约为文法开始符号S的过程
    b. 自顶向下的语法分析采用最左推导方式；自底向上的语法分析采用最左归约方式（反向构造最右推导）
    c. 自底向上语法分析的通用框架：移入-归约分析(Shift-Reduce Parsing)
        i. 在对输入串的一次从左到右扫描过程中，语法分析器将零个或多个输入符号移入到栈的顶端，直到可以对栈顶的一个文法符号串β进行归约为止
        ii. 然后，它将β归约为某个产生式的左部
        iii. 语法分析器不断地重复这个循环，直到它检测到一个语法错误，或者栈中包含了开始符号且输入缓冲区为空(当进入这样的格局时，语法分析器停止运行，并宣称成功完成了语法分析)为止
    d. 移入-规约分析的动作
        i. 移入：将下一个输入符号移到栈的顶端
        ii. 归约：被归约的符号串的右端必然处于栈顶。语法分析器在栈中确定这个串的左端，并决定用哪个非终结符来替换这个串
        iii. 接收：宣布语法分析过程成功完成
        iv. 报错：发现一个语法错误，并调用错误恢复子例程
    e. 移入-规约分析的特点
        i. 若栈从左往右为从底到顶，输入从左到右为从顶到底，则 栈+输入 永远=规范句型
        ii. 句柄：每次规约的符号串，应当是句型的最左直接短语，否则容易出错
2. LR分析法：从左扫描输入，反向构造出最右推导序列
    a. LR文法(Knuth, 1963)是最大的可构造出相应移入-归约语法分析器的文法类
    b. LR(k)分析：需要向前查看k个输入符号的LR分析
        i. k=0 和 k=1 这两种情况具有实践意义，当省略(k)时，默认k=1
    c. 基本原理：逐步形成句柄，用移进s、待约、规约r等“状态”表示句柄识别进展程度
    d. LR分析器（自动机）中除了符号栈还需要状态栈，对应的，分析表也需要（针对终结符的）动作表和（针对非终结符的）状态转移表，分析表中每行是当前状态，每列是处理中的符号
        i. 初始化：状态栈：s0。符号栈：$。输入区：a1a2…an$
        ii. 平常态：s0s1… sm。$X1…Xm。aiai+1…an$
        iii. ACTION[sm,ai]=**sx**后：s0s1… sm**sx**。$X1…Xm**ai**。ai+1…an$
        iv. ACTION[sm,ai]=rx后(产生式x**A→Xm-(k-1)**…Xm)：s0s1… **sm-k**。$X1…**Xm-k A**。ai+1…an$（从ii.平常态转移过来）
            1）GOTO[sm-k,**A**]=**y**后：s0…sm-k **y**。$X1…Xm-k **A**。ai…an$（紧接着

规约后，从iv.规约态转移过来)

v. 如果ACTION[sm , ai]=acc，那么分析成功

vi. 如果ACTION[sm , ai]=err，那么出现语法错误

输入：串w和LR语法分析表，该表描述了文法G的ACTION函数和GOTO函数

输出：如果w在L(G)中，则输出w的自底向上语法分析过程中的归约步骤；否则给出一个错误指示

方法：初始时，语法分析器栈中的内容为初始状态s0，输入缓冲区中的内容为w$。然后，语法分析器执行下面的程序：

```
令a为w$的第一个符号;
while(1) { /* 永远重复*/
        令s是栈顶的状态;
        if ( ACTION [s，a]= st ) {
                将t压入栈中;
                令a为下一个输入符号;
        } else  if (ACTION [s，a]=归约A→β) {
                从栈中弹出｜β｜个符号;
                将GOTO[t，A]压入栈中;
                输出产生式A→β;
        } else  if (ACTION [s，a]=接受)
                break；/* 语法分析完成*/
        else
                调用错误恢复例程;
}
```

3. LR(0)分析
   a. LR(0)项目/项目：右部的某位置有源点的产生式称为该文法的一个LR(0)项目
      i. 项目描述了句柄识别的状态
      ii. 右部有k个符号的项目有k+1个项目
         1) 例：S→bBB的项目：
            a) 移进项目：S → ·bBB
            b) 待约项目：S → b·BB和S → bB·B
            c) 规约项目：S → bBB·
         2) 例：产生式A→ε只生成一个项目A→ ·
   b. 后继项目（Successive Item）：同属于一个产生式的项目，但圆点的位置只相差一个符号，则称后者是前者的后继项目
      i. GOTO(I,X)=CLOSURE({A→αX·β | A→α·Xβ∈I})
      ii. 例：A→α·Xβ的后继项目是A→αX·β
   c. 项目集闭包(Closure of Item Sets)：把等待相同非终结符或其对应首符的等价项目组成集合
      i. CLOSURE(I) = I∪{B→ ·γ | A→α·Bβ∈CLOSURE(I), B→γ∈P}
      ii. 每个项目集闭包对应着自动机一个状态
   d. 增广文法(Augmented Grammar)：如果G是一个以S为开始符号的文法，则G的增广文法G'就是在G中加上新开始符号S'和产生式S'→S而得到的文法
      i. 引入这个新的开始产生式的目的是使得文法开始符号仅出现在一个产生式的左边，从而使得分析器只有一个接受状态
   e. LR(0)自动机：基于增广文法，分析每个项目集闭包的后继项目，生成分析表
      i. LR(0)自动机的规范状态集/项集族 (Canonical LR(0) Collection)：C={I0}∪{I | 存在J∈C, X∈V_N∪V_T, I=GOTO(J,X) }

ii. 每条项目集闭包之间的转移边对应分析表的一行action s或GOTO
iii. 对应增广文法的新产生式的项目集闭包所对应行只有一个表项，是$列的action（acc）
iv. 其他每个没有后继项目的项目集闭包在分析表中没有GOTO，但每个action都是r该项目集对应的产生式
v. 给定文法G = ( VN, VT, P, S)，对应LR(0)自动机 M = ( C, V_N∪V_T , GOTO, I0 , F)。其中状态集C={I0}∪{I | 存在J∈C, X∈V_N∪V_T, I=GOTO(J,X ) }；初始状态I0=CLOSURE({S'→.S })；终止状态F={ CLOSURE({S' →S.}) }

```
SetOfItems CLOSURE(I) { //计算给定项目集I的闭包
    J=I;
    repeat
        for ( J中的每个项A →α·Bβ)
            for ( G的每个产生式B→γ)
                if (项B→· γ不在J中)
                    将B→· γ加入J中;
    until 在某一轮中没有新的项被加入到J中;
    return J;
}
SetOfItems GOTO(I，X ) { //返回项目集I对应于文法符号X的后继项目集闭包
    将J初始化为空集;
    for ( I中的每个项A→α·Xβ)
        将项A→αX·β加入到集合J中;
    returnCLOSURE (J);
}
void items( G' ) { //求项目集闭包
    C = { CLOSURE ({[S'→ ·S] } ) };
    repeat
        for(C中的每个项集I)
            for(每个文法符号 X )
                if( GOTO( I，X)非空且不在C中 )
                    将GOTO( I，X)加入C中;
    until在某一轮中没有新的项集被加入到C中;
}
//LR(0)分析表构造算法
构造G'的规范LR(0)项集族C = { I0, I1, … , In }
令Ii对应状态i。状态i的语法分析动作按照下面的方法决定:
if A→α·aβ∈Ii and GOTO( Ii, a )=Ij then ACTION[ i, a ]=sj
if A→α.Bβ∈Ii and GOTO( Ii, B )=Ij then GOTO[ i, B ]=j
if A→α·∈Ii and A ≠ S' then for a∈V_T∪{$} do ACTION[ i, a ]=rj （j是产生式A→α的编号）
if S'→S·∈Ii then ACTION [ i, $ ]=acc
没有定义的所有条目都设置为“error”
//若按上述算法判断action时出现移进/规约冲突或规约/规约冲突（即一个元素多种动作），说明该CFG不是LR(0)文法
```

4. SLR分析（Simple的LR冲突化解方案）
    a. 解决冲突的思想
        i. 设移进项目形如Ai→αi·aiβi；规约项目形如Bi→γi·。若FOLLOW(Bi)两两不相交，且都不包含任一ai，则按以下原则解决:
        ii. 输入符号a是ai中的一个，则移进a;
        iii. 输入符号a属于FOLLOW(Bi)，则规约成Bi→γi;

iv. 否则报错

b. 构造算法：

构造G'的规范LR(0)项集族C = { I0, I1, … , In}

根据Ii构造得到状态i。状态i的语法分析动作按照下面的方法决定：

if A→α·aβ∈Ii and GOTO( Ii , a )=Ij then ACTION[ i, a ]=sj

if A→α.Bβ∈Ii and GOTO( Ii, B )=Ij then GOTO[ i, B ]=j

if A→α·∈Ii and A ≠ S' then for **a∈FOLLOW(A)** do ACTION[ i, a ]=rj（j是产生式A→α的编号）

if S'→S·∈Ii then ACTION[ i, $]=acc;

没有定义的所有条目都设置为“error”

//与传统LR区别是action[i,a]=rj的条件缩减了。若按上述算法判断action时出现移进/规约冲突或规约/规约冲突（即一个元素多种动作），说明该CFG不是SLR文法。（如FOLLOW集有运算符和$时依旧不知道是移入还是规约好）

5. LR(1)分析

a. 解决冲突的思想

i. SLR中，输入符号属于FOLLOW(Bi)只是规约的必要条件，而非充分条件

ii. LR(1)项：形如[A→α·β, a]的项，其中A→αβ是一个产生式，a是一个终结符(这里将$也视为一个特殊的终结符)它表示在当前状态下，A后面必须紧跟的终结符，称为该项的展望符(lookahead)

1) 当β不是ε，即**•不在产生式最右时，展望符并没有作用**

2) 当β是ε，即**•在产生式最右时，只有输入展望符a才能规约A→α**

3) a的集合总是FOLLOW(A)的子集，而且一般是真子集

iii. 等价LR(1)项：形如A→α·Bβ, a的项，有等价项B→·γ, b

1) 其中b与产生式B→γ无关，只与β和a有关，即b∈FIRST(βa)

2) 当β =>^+ ε时，b=a称作继承的后继符，否则叫自生的后继符

项目集闭包CLOSURE( I ) = I∪{ [B→·γ,b] | [A→α·Bβ, a] ∈CLOSURE( I ), B→γ∈P,b∈FIRST(βa)}

```
SetOfItems CLOSURE( I ) {
        repeat
                for ( I中的每个项[A→α·Bβ，a])
                        for (G'的每个产生式B→γ)
                                for (FIRST (βa)中的每个符号b)
                                        将[ B→· γ ，b]加入到集合I中;
        until不能向I中加入更多的项;
        until I ;
}
```

后续集闭包GOTO( I, X ) = CLOSURE({[A→αX·β,a]|[A→α·Xβ, a]∈I})

```
SetOfItems GOTO( I，X ) {
        将J初始化为空集;
        for ( I中的每个项[A→α·Xβ，a] )
                将项[ A→αX·β，a ]加入到集合J中;
        return CLOSURE (J);
}
```

构造文法G'的LR(1)项集族

```
void items(G') {
        将C初始化为{CLOSURE ({[S'→· S, $]} )} ;
        repeat
                for ( C中的每个项集I )
                        for ( 每个文法符号X )
                                if ( GOTO( I，X )非空且不在C中 )
                                        将GOTO( I，X )加入C中;
        until 不再有新的项集加入到C中;
}
```

LR分析表构造算法
构造G'的规范LR(1)项集族C = { I0, I1, … , In}
根据Ii构造得到状态i。状态i的语法分析动作按照下面的方法决定：
if[A→α·aβ,b] ∈Ii and GOTO( Ii , a )=Ij then ACTION[ i, a ]=sj
if[A→α·Bβ,b] ∈Ii and GOTO( Ii , B )=Ij thenGOTO[ i, B ]=j
if[A→α·, **a**] ∈Ii and A ≠ S' then ACTION[ i, a ]=rj （j是产生式A→α的编号）
if[S'→S·,$] ∈Ii then ACTION[ i, $]=acc;
没有定义的所有条目都设置为“error”
若按上述算法判断action时出现移进/规约冲突或规约/规约冲突（即一个元素多种动作），说明该CFG不是SLR文法

6. LALR(1)分析（LookAhead-LR）
    a. 节省状态数的思路
        i. 同心项目集：除了展望符外都相同的项目集
        ii. 将同心项目集合并（指让展望符变成展望符集合，常用/分割），若分析表没有冲突，称该文法为LALR(1)文法
        iii. 合并后的展望符集合仍为FOLLOW集的子集
    b. 冲突
        i. 合并后可能产生规约-规约冲突（指合并前，不同状态里同样的符号栈遇到同样的输入符可能使用不同产生式规约）
        ii. 合并后不会产生移进-规约冲突（因为展望符只和规约有关）
        iii. 合并后可能会使错误的发现被推迟（因为LALR可能会被合并后的展望符指引到多余的规约动作）
    c. 特点
        i. 形式上与LR(1)相同
        ii. 规模上和LR(0)/SLR相当
        iii. 分析能力：SLR<LALR(1)<LR(1)（有错，但错误不多）
7. 二义性文法的LR分析
    a. 二义性文法=有冲突的文法=不是LR文法
    b. 因为消除二义性要引入新的产生式，所以二义性文法一般更简短、自然
    c. 利用运算符优先级和结合性可解决一些冲突（如加号乘号，如if else）
    d. 必须在严格控制下才能保守使用二义性文法，因为稍有不慎就会让语法分析器所识别的语言出现偏差
8. LR分析的错误处理
    a. 可能的出错时机：查表发现空项
    b. 恐慌模式错误恢复：从栈顶向下扫描，直到发现某个状态si，它有一个对应于某个非终结符A的GOTO目标，可以认为从这个A推导出的串中包含错误
        i. 然后丢弃0个或多个输入符号，直到发现一个可能合法地跟在A之后的符号a为止
        ii. 之后将si+1=GOTO(si , A)压入栈中，继续进行正常的语法分析
    c. 短语层次错误恢复：检查LR分析表中的每一个报错条目，并根据语言的使用方法来决定程序员所犯的何种错误最有可能引起这个语法错误，然后构造出适当的恢复过程（将分析表的空白项填成对应的语法错误信息）